AFRL-IF-RS-TR-2004-91
Final Technical Report
March 2004

SKEPTICAL SYSTEMS

Honeywell ACS Laboratories

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. N124

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-91 has been reviewed and is approved for publication

APPROVED: /s/
THOMAS M. BLAKE
Project Engineer

FOR THE DIRECTOR: /s/
WARREN H. DEBANY, JR.
Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2004	**FINAL** **May 02 – Apr 03**

4. TITLE AND SUBTITLE

SKEPTICAL SYSTEMS

5. FUNDING NUMBERS

C - F30602-02-C-0116
PE - 63760E
PR - N124
TA - 96
WU - 10

6. AUTHOR(S)

Christopher W. Geib

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Honeywell ACS Laboratories
3660 Technology Drive
Minneapolis MN 55418-1006

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFGB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-91

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Thomas M. Blake/IFGB/(315) 330-1482 Thomas.Blake @rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This report describes the technical components developed for hostile intent recognition, and documents specific experiments to evaluate a demonstration skeptical system in a cyber security domain. Current systems execute any command issued by an authenticated user so long as they fall within the privileges granted. By contrast, a skeptical system entertains doubts about the tasks that it is asked to perform. It questions the authenticity, integrity, and intent of the requester and acts with due consideration of its doubts. The report is broken down into three major sections. The first section discusses the system architecture overview and the technologies behind the demonstration system. The next section discusses a number of demonstration scenarios and the system's performance on them. The third section discusses scalability experiments done on the intent recognition algorithm (one of the central components of the system). The report closes with general conclusions about the system and makes some suggestions for areas for future work.

14. SUBJECT TERMS Skeptical Systems, continued assessment of apparent intent, impact and integrity and authenticity, hostile intent recognition, threat assessment, response planning, user classification and anomaly detection, hostile intent recognition and authentication, Probabilistic Hostile Agent Task Tracker (PHATT) , Bayesian threat evaluator, intent recognition algorithm

15. NUMBER OF PAGES 29

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures and Tables

Introduction

Firewalls, cryptography, intrusion detection, and network management are all valuable in the defense of computer controlled systems. However these technologies are directed at preventing *outsiders* from doing harm to the protected assets. They do not address the threat from *insiders*. These already authenticated users can sidestep many, if not all, conventional security measures. Critical assets need a further layer of security to protect them from hostile agents that have already breached the walls. To address this, we have been working on *Skeptical Systems*.

Current systems execute any commands issued by an authenticated user so long as they fall within the privileges granted. By contrast, *a skeptical system entertains doubts about the tasks that it is asked to perform. It questions the authenticity, integrity, and intent of the requester and acts with due consideration of its doubts*. Skeptical system features can be embedded in a protected asset, or wrapped around it as a guardian. Skeptical Systems are not a technology per se, but a philosophy of interaction that grants considerable autonomy to the protected system under certain conditions. Some existing systems already embody this stance in limited ways. We intend to take the skeptical stance several steps further, including the following novel aspects:

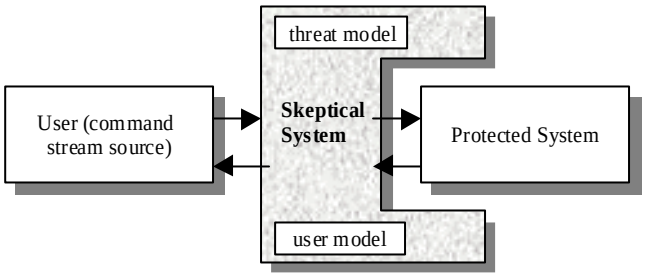

Figure 1. Skeptical System

- the continuous assessment of apparent *intent* behind a command stream through probabilistic task tracking
- the continuous assessment of the *impact* of commands based on application models
- the continuous assessment of the *integrity and authenticity* of the command issuer through user modeling
- *graded responses* to threat based on level of skepticism, proximity of threat

This paper describes some of the technical components we have developed for hostile intent recognition, and documents specific experiments to evaluate a demonstration skeptical system in a cyber security domain. For this demonstration we selected, as an application for protection, Honeywell Labs, in house, Contract Management System (CMS) seen in Figure 2. The CMS is an application used by contract officials and contract managers within the company for creating and monitoring the execution of contracts and their deliverables. It is used to track all of the information critical to each contract including: deliverables, schedules, contact personnel, shipping addresses, and other information for all contracts.

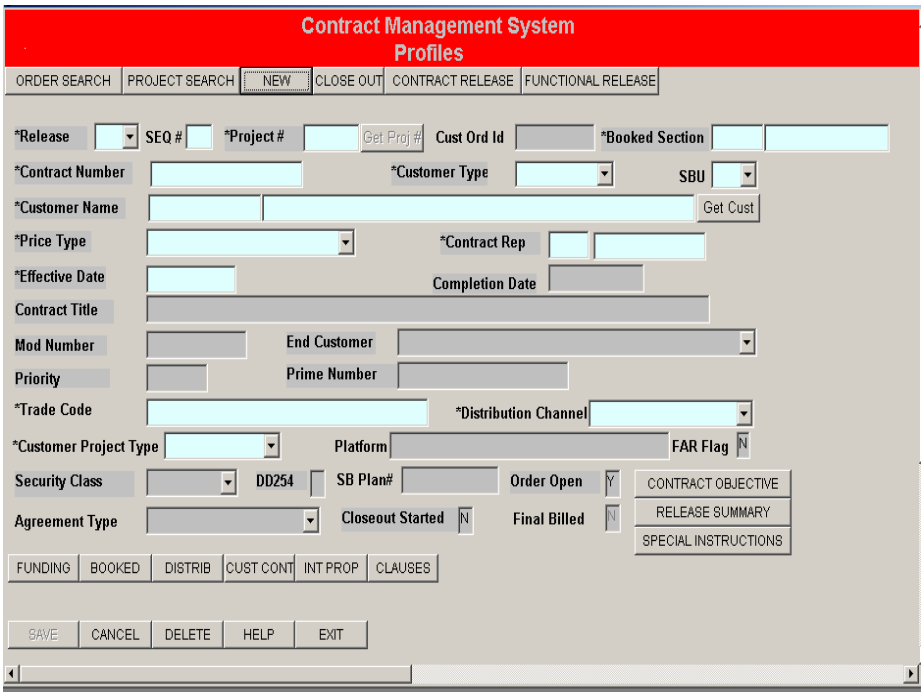

Figure 2. The Contract Management System user interface.

As the central repository for all contract information, if an insider wanted to cause significant disruption to our work, the CMS would be an obvious choice to attack. An insider could gain valuable information about Honeywell Labs contracts from this source to sell to outsiders. Further, contracts could be modified to change when, and where deliverables are sent. Deliverable and even whole contracts could be deleted from the system. The objective of our skeptical system is to demonstrate the ability to recognize and prevent these sorts of actions on the CMS with minimal impact on legitimate users.

The rest of this paper is broken down into three major sections. The first section will discuss the system architecture overview and discussion of the technologies behind our demonstration system. The next section will discuss a number of demonstration scenarios and the system's performance on them. The third section will discuss scalability experiments done on the intent recognition algorithm (one of the central components of the system). The report will close with general conclusions about the system and make some suggestions for areas for future work.

System Architecture

In general the architecture for our implemented skeptical system has five components. The architecture is depicted in Figure 3. The intent recognition component, user classification and anomaly detector, and the authentication modules basically function as sensors. They each take in a record of the users actions and inputs and produce, a belief about the users intentions, a belief about the "normality" of the actions the user is performing, and a belief about the user's identity respectively.

The Threat Assessment module combines the inputs from these three modules and produces an evaluation of the threat represented by the user's actions to the system. On the basis of this assessment the Response Planner can then choose to either execute the actions the user has called for or call for other actions. Note that in some cases user actions can directly trigger actions by the Response Planner. This is designed to cover the case of actions that are so obviously hostile to the system's integrity that they call for immediate action.

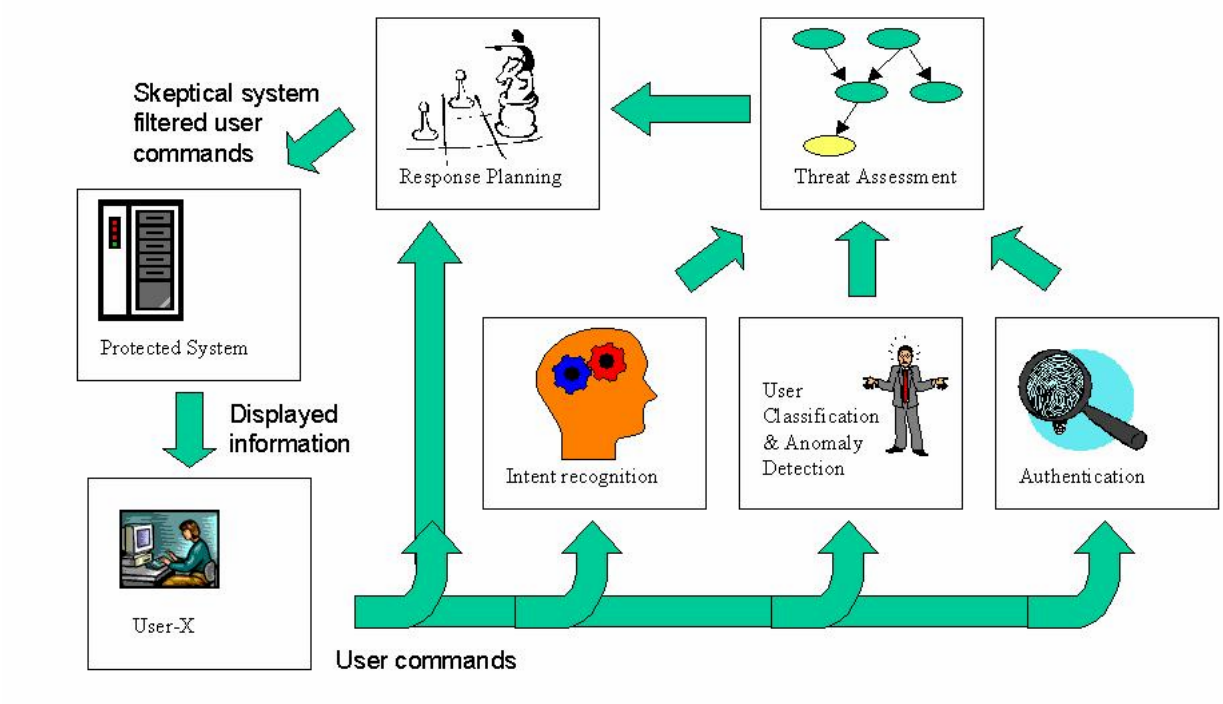

Figure 3. Skeptical System architecture.

The following subsections will provide more detail on how each of the modules of the Skeptical System architecture works.

Technical Approach to Threat Assessment

The threat assessment module combines models of individual users with online information about activity observed by the detection modules. The result is represented within a Bayesian belief

network, which is dynamically updated by new observations (Jensen, 1996). Some of the nodes in this network are potentially observable—when observations are obtained, particular variables are clamped to observed values. Other nodes are never directly observable, so belief in the possible states is computed as a function of the observed nodes and prior beliefs.

Although treated as a single entity, the threat assessment belief network consists of two sections: one for *user models*, and one for *session models*. A session corresponds to a single login session for the CMS application, starting with an attempt to authenticate and ending with a logout or other program termination. The user model section consists of a set of beliefs about the motivations and operating habits of each user known to the system; these models persist across sessions. A simplified example is shown in Figure 4.

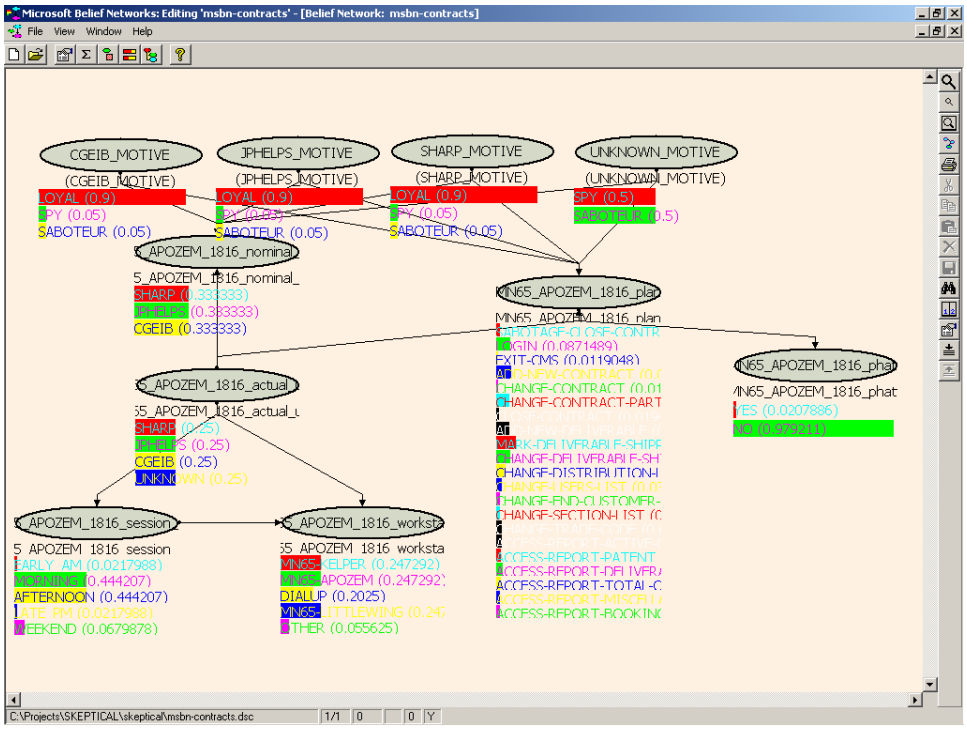

Figure 4. Bayesian belief network generated for modeling user threat.

The user model section has two roles. First, it assigns a belief to various "motives" for each user. In our simple example, the domains of the motives are "loyal", "spy" or "saboteur". More refined characterizations are possible, but the intention is to capture the concept of motivation at a high enough level to encompass a broad range of plans. The true user motivations are always unknown, but may be inferred. A knowledge base maintains prior information about the users and possible classes of motives. User models are assigned to real people, as opposed to login identities—a real person may assume multiple identities when interacting with the computer. In addition to the known set of registered users, there is an "unknown user". The unknown user is a real person who is not registered, perhaps because he/she is an interloper, e.g. an outside hacker or, an insider that is not normally authorized to use the system. The unknown user is represented as a single individual in our prototype, but there might be zero or more than one such person, operating independently or

in conspiracy; handling this is an area of interest for future research. Initially, prior belief designates all uses as highly likely to be loyal, except for the unknown user, whose motives are automatically suspicious. An additional part of the knowledge base describes the relation between motives and likely plans. For example, a user with a strong "spy" motive is unlikely to engage in plans whose goals are primarily destructive (sabotage). Insiders are however assumed to continue to have motivation to carry out normal activities, i.e. to carry out regular work assignments. A second role of the user model section is to characterize user habits. This is described in the section User Classification and Anomaly Detection.

Session models are built dynamically by the threat assessment module as users authenticate. Each new session generates a unique session identifier and this event causes a new subgraph to be added to the threat assessment belief network. One portion of this new subgraph relates to the identity of the user, and another portion to the goals of the session. The session model subgraph is connected to the user model, since user motives and habits will causally affect the activities of sessions. A distinguished node in the session network connects the true (unobservable) root goal for the session to the PHATT-reported goal for the session (observable). This "true goal" node is the principal product of the threat assessment system passed to the response planning system. It reflects the net assessment of both the user motives and of the PHATT plan recognition system.

Technical Approach to Response Planning

The skeptical response planner is a simple, fast, and efficient mechanism for effecting a graded response to insider threat. It combines inputs from PHATT, a Bayesian threat evaluator, a response state model, and a response action ontology to determine the best response to apply to a given situation. The collection of preconditions and associated response instantiation definitions is called the response library. Although called a "planner," the response planner does not presently plan beyond the next applicable action(s) to a given situation since its goal is simply to preserve a safety level for the application within the current application context.

The response planner is an integral part of a skeptical system instantiation, so it communicates with other skeptical system components through shared memory. It communicates the appropriate response to its host application, when appropriate, through an XML stream implemented on a socket layer.

The skeptical response planner selects the best response available from its response library for a given situation and then signals its host application to apply the response. To select the best operator, the planner tests the applicability of each response in the response library. If the response is applicable, its cost is calculated – cost is a unitless measure of relative utility, where lower cost is better. If there exists an applicable, lowest cost action, it is selected and then applied.

Operator applicability is tested using the operator's preconditions. The preconditions are established in the operator definition. The operator preconditions are structured as *if …elseif* structures. This means that there may be arbitrarily many mutually exclusive situations in which an operator may apply; additionally, each situation may parameterize the resulting operator instantiation differently.

Each test contained within the precondition's *if…elseif* structure may be an arbitrarily complex test of a given scenario as identifiable through queries to PHATT, the Bayesian threat evaluator, the response state mode, and the action ontology. This enables a graded response to a large number of action sequences, which precise control over the precondition test granularity. The natural way to model the graded response from one scenario to another is as a finite state machine (FSM) where each class of scenarios is represented by part of the FSM, and where each scenario in a class is represented as a state in its FSM. There are typically a number of synonymous motive state changes that will switch evaluation from one part of the FMS to another.

Figure 5 diagrams a possible transition in the FSM. For a given set of threat evaluation motives and plan intentions, which represent the states in the FSM, there are transitions for new plan intentions and action classes.

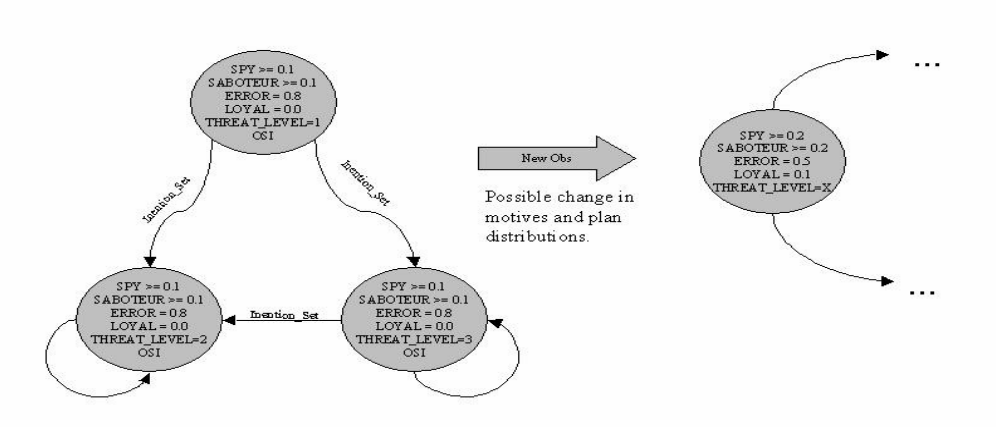

Figure 5. A depiction of the skeptical response planner FSM.

A key question then is at what granularity of change should the response transitions be coded. We found that modeling operator precondition change at a high level of granularity was sufficient to cover threatening scenarios of interest without being too invasive. Modeling precondition test change at a high level of granularity means that only a relatively large change in threat evaluation and PHATT intention identification would push the skeptical response planner into new modes of response. However, there were some situations where there was increasing suspicion by the skeptical system and then a potentially destructive action, where we would want to respond more aggressively. An action ontology provided the necessary instrument to provide that result, where actions were classified as read, change, or delete actions, and where change and delete actions could then be questioned or aborted with increasing but still low levels of user suspicion on the part of the skeptical system.

The responses included in the response library included the following:

- Take a picture of the user
- Lock screen requiring a fingerprint
- Backup and changes made to the database
- Lock the account requiring a password

- Require the CMS password
- Require supervisor authentication
- Notify the supervisor
- Start a session log
- Validate last command (Are you sure you want to X?)
- Fake form/report data
- Warn that the last command is not appropriate
- Notify the system administrator
- Abort the last action
- Terminate the user's session

Technical Approach to User Classification & Anomaly Detection

By design, the skeptical system is never completely certain of the identity of the person directing a given session. A session begins with a *nominal user* requesting to login and presenting a password. The *actual user* may be any of the registered users or the special "unknown" user. The nominal user is always known, but the actual user can only be inferred with greater or lesser confidence. Inferring the actual user from the nominal user (and other evidence) is the job of the user classification and anomaly detection module. This function is tightly integrated into the threat assessment Bayesian belief network, a portion of which is shown in Figure 4.

The skeptical system prototype uses a modular approach to evidence sources for user classification. This allows it to be easily extended as new methods of identification become available. Currently experiments have used the following forms of evidence:

- Nominal user (who the user claims to be)
- Access mode anomaly detection
- Operating system native authentication
- Biometric devices

The access mode anomaly detection approach is based on the workstation from which the user started the session, and the time of day. Part of the user model for each actual user is an estimate of the likelihood that the user will begin a session from a particular workstation at a particular time of day (broken into blocks such as morning, afternoon, evening, weekend). Once a session has started, this classifier yields evidence that implicates one or more actual users. A slow updating process can track changing user habits. The operating system of the user workstation will also typically demand some form of native authentication. The CMS skeptical agent inside the client application will detect the owner of the process, and pass this name to the skeptical system as a further piece of evidence. Finally, the skeptical system may ask for biometric authentication at any time during a session; the result also adds evidence for an actual user. As biometric authentication was not one of the research foci, we used commercially available hardware and software additions in our testbed. Client systems were fitted with a URU fingerprint scanner and a LogiTech camera to take pictures of the current user. Other forms of evidence such as the keystroke timings or mouse motion patterns could readily be incorporated.

Note that the updating of the belief network assigns a posterior distribution to all of the nodes. Evidence implicating a particular actual user may influence the assessment of the likelihood of a certain session goal. For example, a plan to exfiltrate data is more plausible when an individual that is already suspected of spying motives is believed to be the actual user. Conversely, observation of a plan consistent with spying may make a suspected saboteur seem less likely as an actual user.

Technical Approach to Hostile Intent Recognition

Our plan/intent recognition framework PHATT (Probabilistic Hostile Agent Task Tracker) is based on the realization that plans are executed dynamically and that at any given moment the agent is able to execute any one of the actions in its plans that have been enabled by its previous actions. To formalize this, initially the executing agent has a set of goals and chooses a set of plans to execute to achieve these goals. The set of plans chosen determines a set of pending primitive actions. The agent executes one of the pending actions, generating a new set of pending actions from which then next action will be chosen.

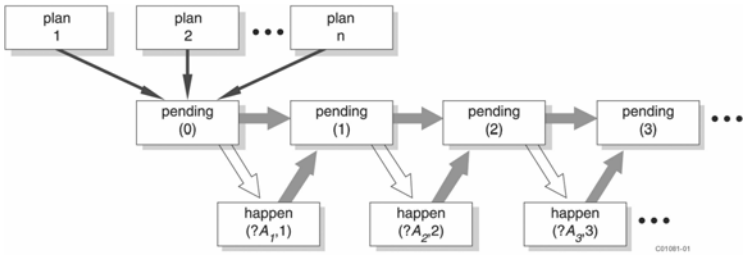

Figure 6. Probabilistic Hostile Agent Task Tracker (PHATT)

Each pending set is generated from the previous set by removing the action just executed and adding newly enabled actions. Actions become enabled when their required predecessors are completed. This process is illustrated in Figure 6. To provide some intuition for the probabilistically-inclined, the sequence of pending sets can be seen as a Markov chain, and the addition of the action executions with unobserved actions makes it a hidden Markov model. There are a number of probabilistic reasoning algorithms that can be used to infer the agent's goals given such a Markov model. We have designed a new particularly efficient algorithm that makes use of the deterministic aspect of much of the problem and only introduces probabilities where necessary. As a result, PHATT is able to handle a number of problems that are critical to real domains including: multiple interleaved goals, partially ordered plans, the effects of context on the goals adopted, the effect of negative evidence or failure to observe ("the dog didn't bark"), missing observations, and observation of "failed actions." See (Geib & Goldman, 2001a,b and 1999) for a more complete discussion of the system its algorithm and these issues. This technology was initially developed in the context of the DARPA CyberPanel sponsored Honeywell Argus project. We are currently applying it to recognizing the activities of daily living of patients in assisted living situations.

The plan libraries divide into coverage of normal and malicious user behavior. Our modeling of normal user behavior was CMS form centric, so each CMS form has a corresponding set of PHATT plans describing normal usage. Since the plans defining malicious usage relied on normal plan definitions, we modeled malicious theories of usage as separate domains – the sabotage and

spying domains. So, for instance, the Contract Management form had four normal usage plans – adding a new contract, changing an existing contract, changing contract particulars, and closing out a contract. The sabotage domain then contained a plan describing a type of sabotage that involved closing several contracts without support for concluding that this was normal behavior.

PHATT's conclusions about user intent are communicated back to a TCL-based Skeptical Monitor that shows PHATT's distribution of belief over the plans in Figure 7.

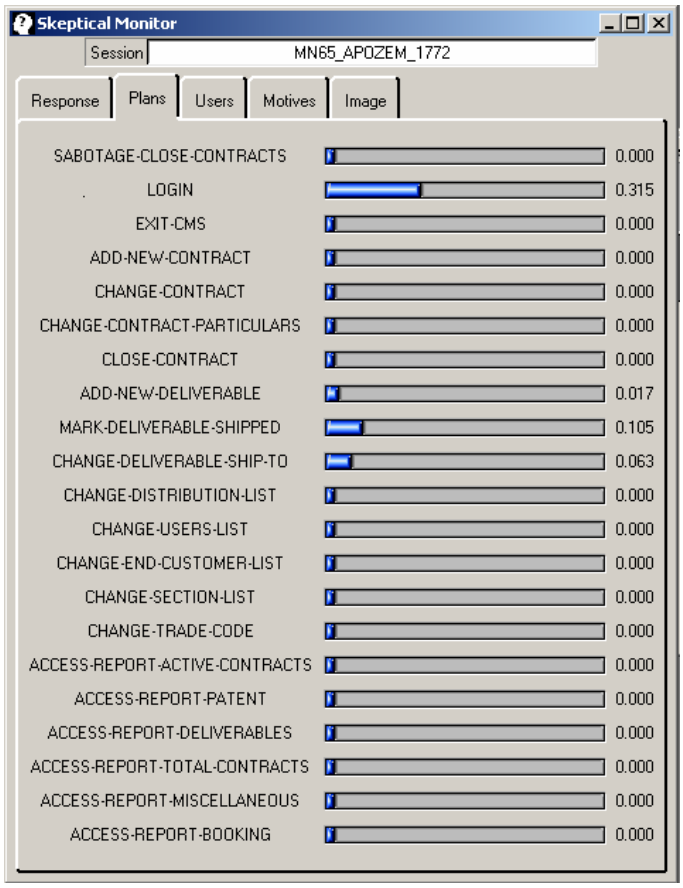

Figure 7. Skeptical Monitor showing PHATT's plan belief distribution.

Basic System Demonstration Experiments

Once our demonstration system was built, we experimented with six test scenarios, covering the major classes of response behavior for the skeptical CMS application. The first scenario covered innocuous behavior, where the user's login attributes and plans never raised significant suspicion. The second scenario covered the case when the user's login was abnormal, including an abnormal login time or abnormal login session ID. The third and fourth scenarios covered sabotage behavior, with the third scenario covering the less overt case of sabotage, and the fourth the more overt case. The fifth and sixth scenarios cover spying behavior, with the fifth covering the less over case of spying, and the sixth, the more over case of spying. In these cases the distinction between spying and sabotage is that spies are only interested in acquiring information while the saboteurs actively modify or delete information from the system.

In the following sections, graphs of the systems performance on each of these test scenarios will be presented. Along with the graphs will be a brief discussion of the important events/points in the scenarios. Each set of graphs will best be viewed in color and will have the following structure.

Graph Descriptions All of the panels are aligned along the sequence of observations.

1. Time: the top panel shows the runtime of the Skeptical System, partitioned by component, for each observation, measured in milliseconds.
2. Response: Shows the system responses at the stage activated.
3. Normal, malicious: Display the system's belief that various normal or malicious plans are being followed by the agent. This is the output of the PHATT component.
4. jphelps: Displays the skeptical system's belief in the high level motives of the nominal user, "jphelps" (loyal, spying, sabotage). Other users are tracked, but not shown.
5. user: Displays the system's belief in the true identity of the user. A user "unknown" is present to accumulate belief in "someone outside our knowledge".

Innocuous

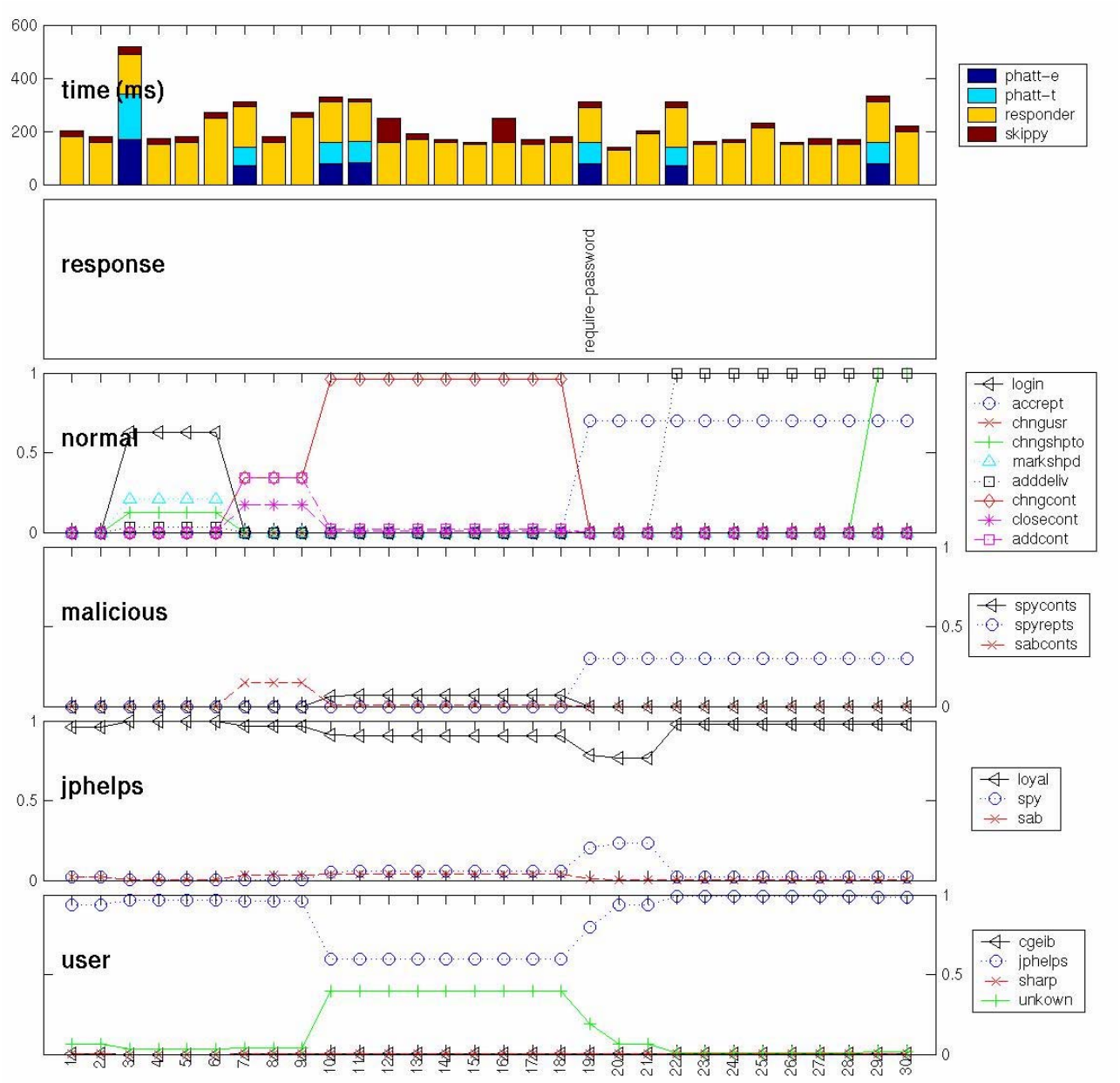

Figure 8. Innocuous

These graphs depict innocuous use. As the user executes individual actions, the system generates observations, which are recorded in a log and displayed in the chart. To conserve chart space, the observations' names have been replaced by numbers. These observation numbers are used as the horizontal axis of the bottom panel. The plan that these actions reveal is:

- Observations 1-6: normal login procedure for the contract management system.
- Observation 7-18: accessing the contracts subform.

- Observations 19: accessing reports (this is a somewhat strange sequence of actions and quite unlikely to occur in this user's normal job function – as codified in the user's default motives and behaviors).
- Observations 20-27: accessing more reports and checking deliverables, but not staying long in any one place; after the user's identity has been revalidated, this barely malicious behavior does not raise the system's concern.

Abnormal NetID

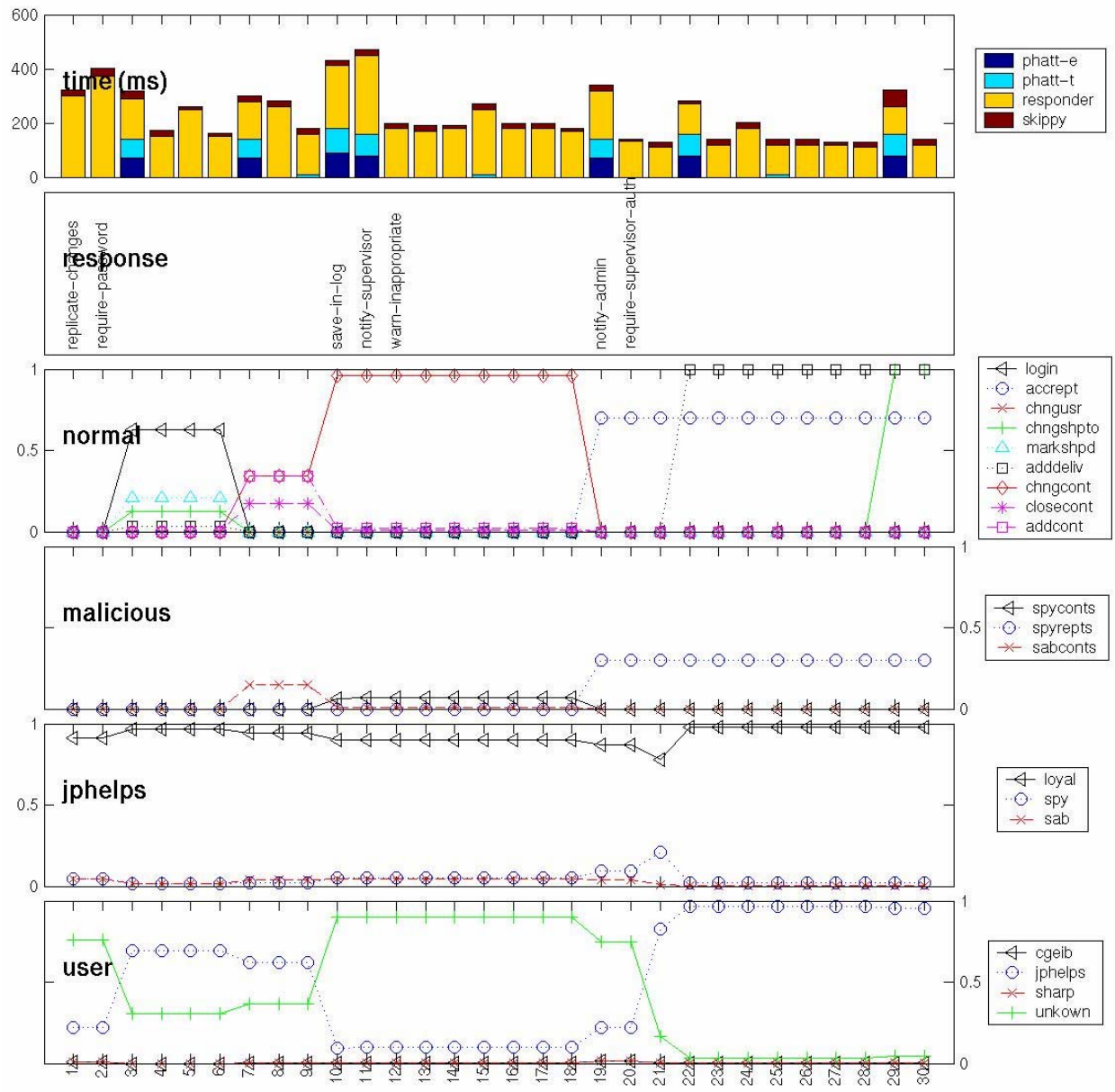

Figure 9. Abnormal NetID

This scenario depicts a user accessing the system with an abnormal network ID. This abnormal network ID reported to the application causes immediate high suspicion of the validity of the user, causing the system to ask for more authentication information. The plan that the user's actions reveal is:

- Observations 1-6: normal login procedure for the contract management system; since the user is under suspicion, the system immediately starts replicating any changes made to the system, in case some of them are destructive. The system also asks for an additional system password to confirm the user's identity.
- Observation 7-9: accessing the contracts subform.
- Observations 10-18: accessing aspects of the contracts subform necessary to change contracts, and then proceedings to change a few contracts. As soon as the already suspect user starts to appear as if he is about to change the contracts, the system's wariness increases, partly because of the two other explanations available to it: that the user may be spying on contract particulars or sabotaging contracts. The increased wariness causes the system to save the actions of the user in a log for forensics and rollback (together with replication). After the system starts logging, the user continues to pursue a course of action that it believes is potentially malicious, especially now that the user's actions are not in the scope of what would normally be expected from the expected user, which drives down the system's belief that the user is who the current authentication information claims. This causes the system to take more immediate precautions against loss. The supervisor is notified of the user's actions, and the user is also finally warned that their actions are not appropriate.
- Observations 19-30: accessing reports in detail causes the system to become very suspicious of the user and the system admin is now notified of potential abuse. Since the user continues with a strange mode of operation, the system finally requires supervisor authentication. Once that is given, the system's belief in the authenticity of the user goes up, but this does not change the belief in the type of plans the user is pursuing. Since the supervisor authentication information is given, the user is allowed to continue using the system. If this information was not given, the user would be effectively locked out.

Saboteur Complete

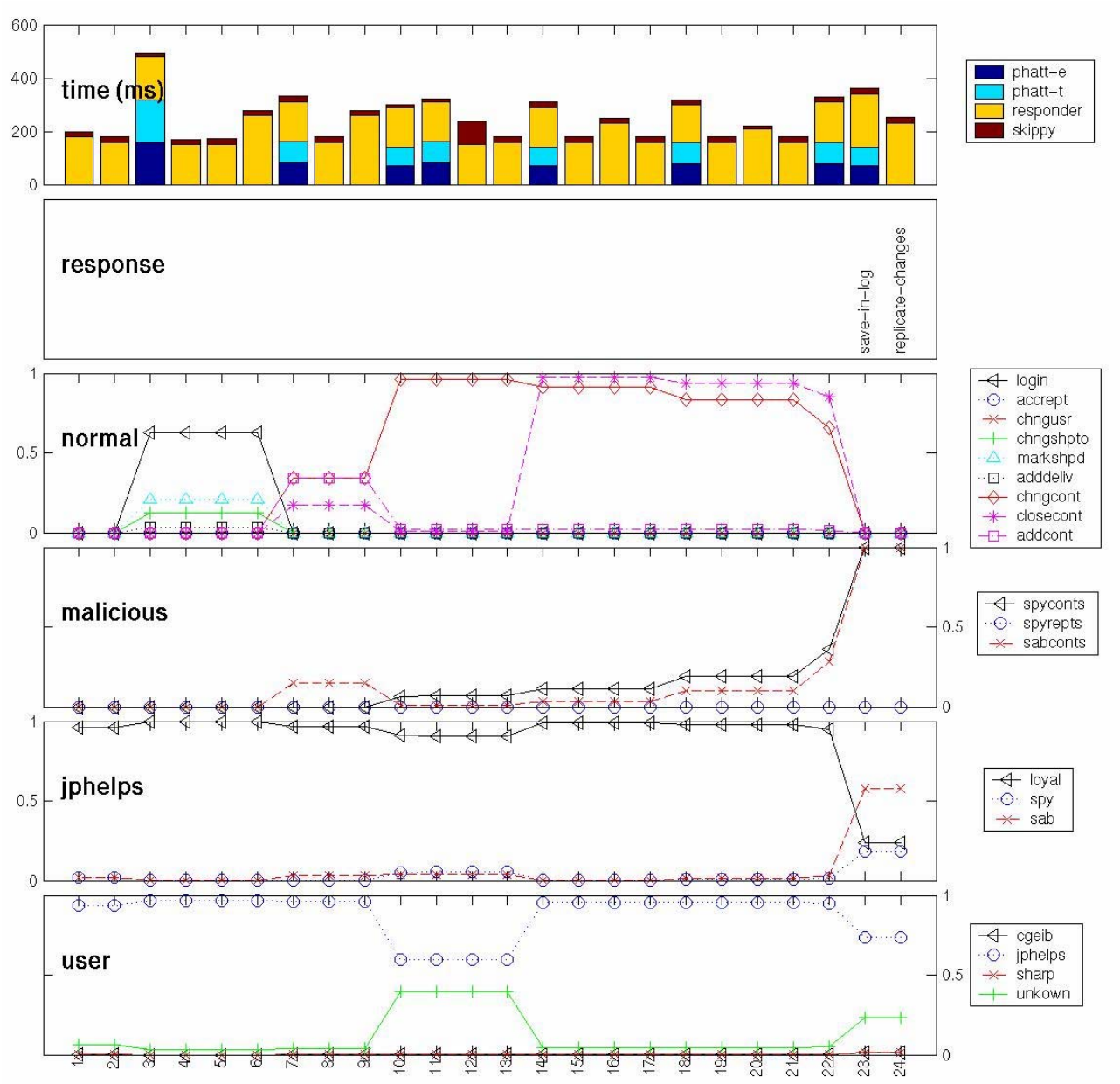

Figure 10. Saboteur Complete

This scenario depicts a user sabotaging the system. The plan that these actions reveal is:

- Observations 1-6: normal login procedure for the contract management system.
- Observation 7-9: accessing the contracts subform.
- Observations 10-22: accessing contracts, including contract details; the user also starts closing out contracts. This causes the contract information to be lost.

- Observations 23-24: contract sabotage – belief in this plan type is driven up by repeated contract closing with no other explanation for the behavior. All other aspects of the user's identify are not in question, but the pursuit of this plan causes the system to become suspicious of the user. The pursuit of a malicious plan causes the system to believe the user's actual identity is unknown. It responds by logging all of the user's actions and replicating any changes made.

Saboteur Incomplete

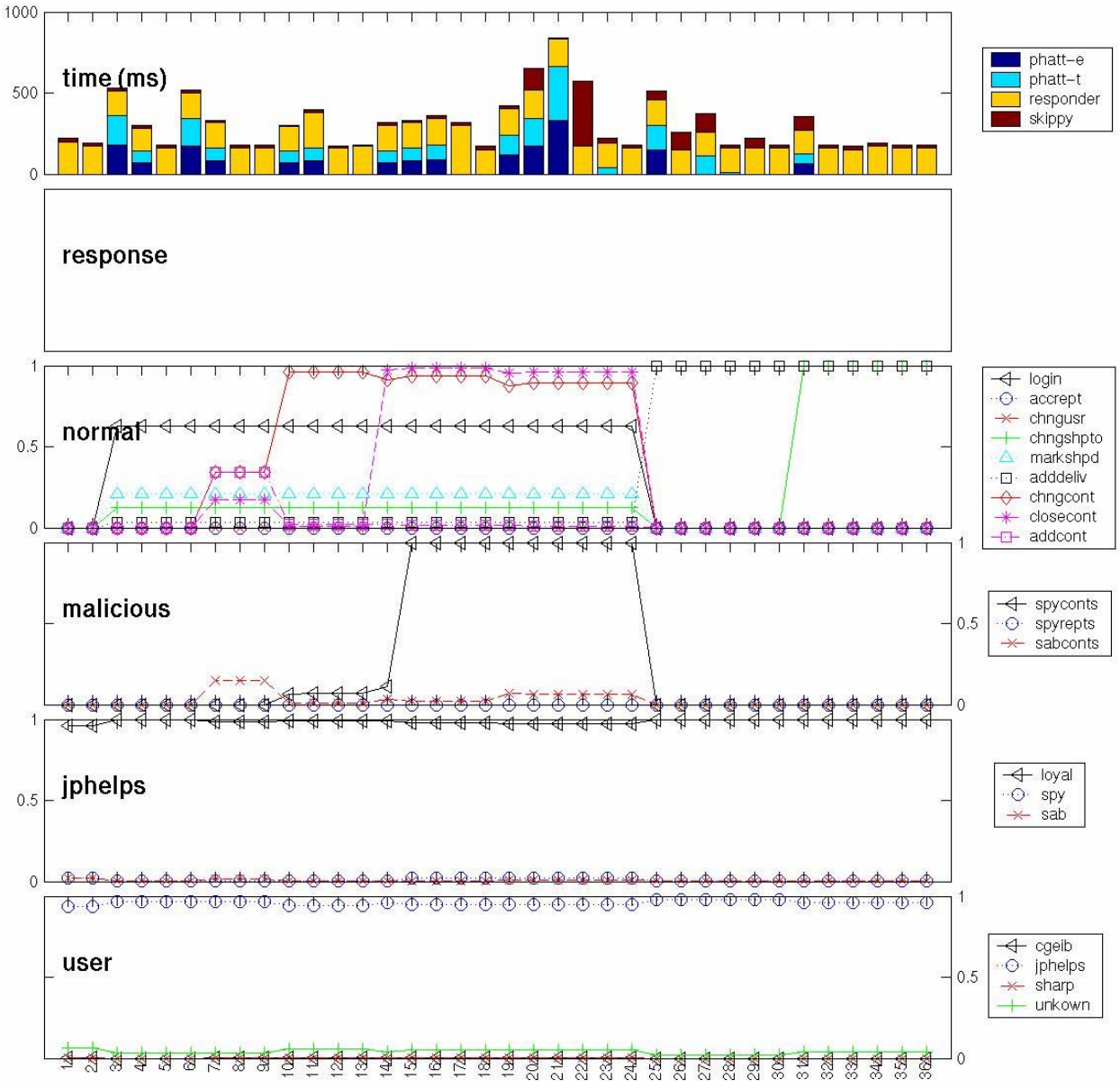

Figure 11. Saboteur Incomplete

These graphics depict system states when a user comes one step away from executing a sabotage plan. The plan that these actions reveal is:

- Observations 1-6: normal login procedure for the contract management system (the system fails to record a login closing action since it was probably malformed, so PHATT keeps this plan open).
- Observation 7-36: accessing reports, closing two contracts, adding a deliverable, changing a deliverable's ship-to address. All of which are considered nominal user actions in moderation.

Spy Complete

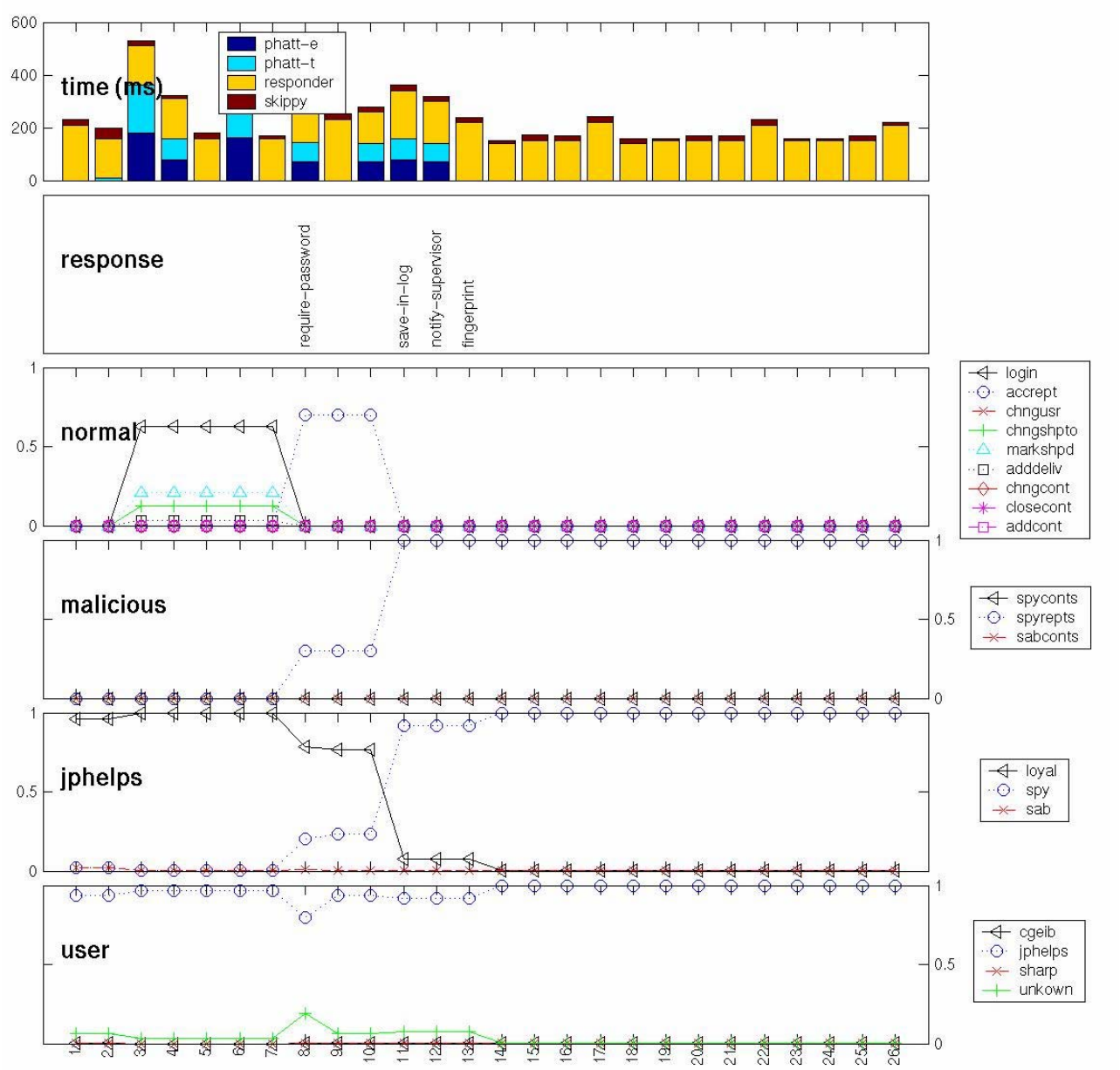

Figure 12. Spy Complete

These graphs depict a user spying on the system. A complete spying plan is pursued and the system responds by attempting to authenticate the user with higher and higher confidence. The plan that these actions reveal is:

- Observations 1-7: normal login procedure for the contract management system.
- Observation 8: accessing the report submenu.
- Observations 9-19: accessing reports (the sequence of reports requested by this individual is highly suspicious, and is recognized as a specific espionage plan--quite unlikely to occur in this user's normal job function).
- Observations 20-21: exiting the application.

Spy Incomplete

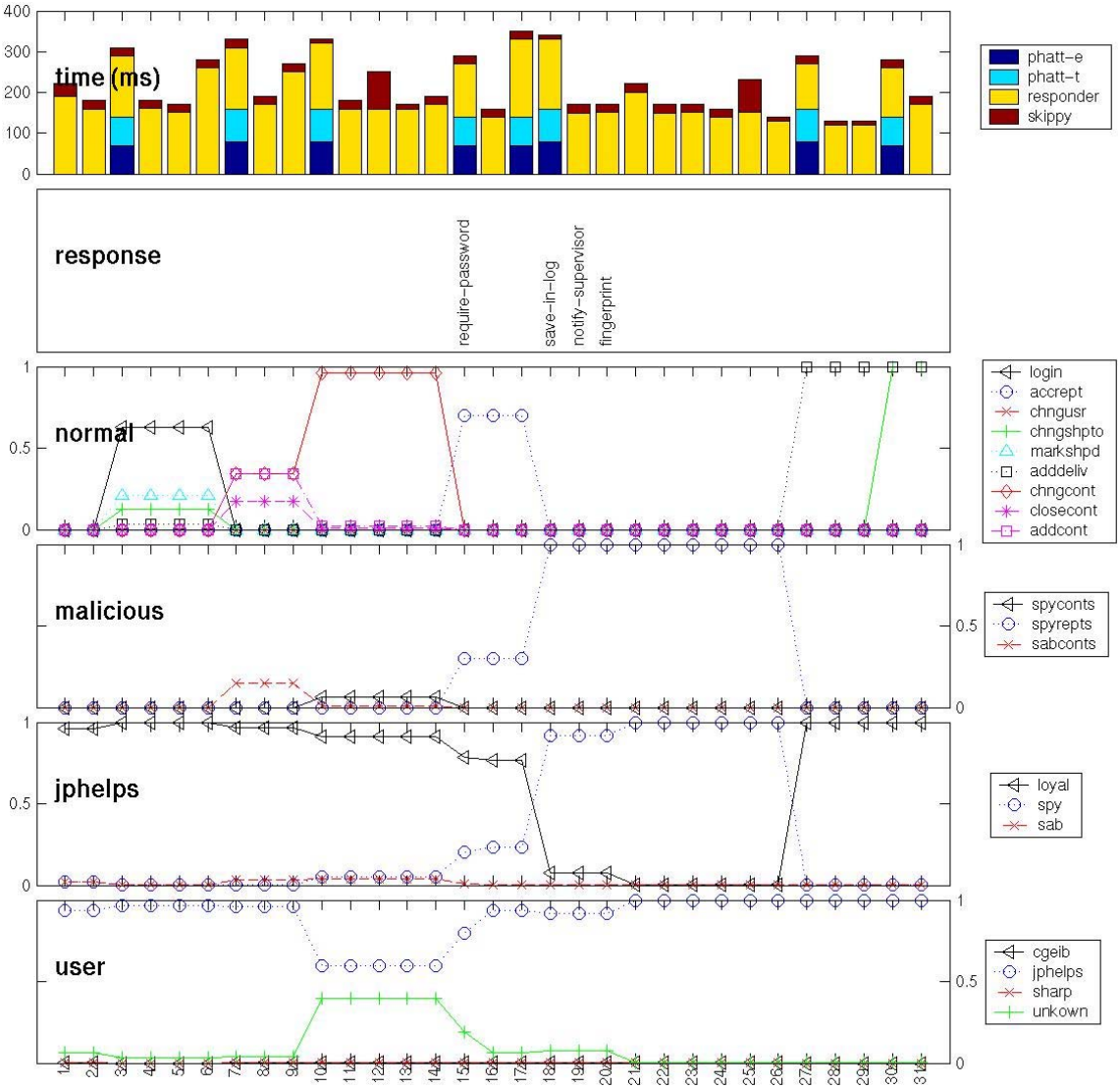

Figure 13. Spy Incomplete

These graphs depict a user that comes one step away from executing a sabotage plan. The plans that these actions reveal is:

- Observations 1-6: normal login procedure for the contract management system.
- Observation 7-9: accessing the contracts manipulation submenu.
- Observations 10-14: changing contract particulars – requires repeated searching of contract particulars – actions which overlap with contract spying.
- Observations 15-31: accessing reports – PHATT concludes that, although the spying plan is not yet complete, the user is in fact spying.

Observations

System Runtimes: Notice that the processing for each of the observations takes no more 600ms and in most cases less than 400ms. The high cost of the response engine is attributable to the particular Bayesian belief network engine we are using for threat analysis, which is currently performing some redundant calculations. We believe this can be significantly reduced. In any case, the time added to the system's response time by the skeptical system will not generally be perceptible to end users.

Issues

MSBNx Most seriously, just loading a relatively simple skeptical threat network in MSBNx, one containing only seven users, and then attempting to assess the probabilities on the PHATT node consistently crashes MSBNx with an "Out of Memory" error on a 1.0GHz PIII with 256MB RAM.

Also of concern, however, is the noticeable system slowdown in Skeptical CMS components when the number of users in the MSBNx network exceeds four. PHATT is hit hardest by MSBNx's resource consumption. Its average observation compute time increases from 924.286 msec to 4017.143 msec for six to seven users in the threat model.

This problem can be mitigated by using a better Bayes Net evaluation product and by user aggregation. We believe that instead of modeling each use, a composite user could be used to model all "unexpected" users during a session.

PHATT Algorithm Scalability Experiments

We have also conducted some initial experiments designed to allow use to understand the most critical factors determining the runtime of the PHATT algorithm. The scalability of the PHATT algorithm to real-world sized domains is necessary for the effective application of this technology. Therefore it is critical that we have a better understanding of the effects that various facets of the plan library will have on the algorithm's runtime. Our initial hypothesis was that while intuitively the number of roots in the plan-library might be assumed to have a large effect on the runtime of the algorithm that in fact other features of the plan library would have more impact. The experiments were also designed to be a starting point for more sophisticated analyses to follow.

Experimental Design

This experiment measures the computation time for the PHATT algorithm as a function of three factors of interest. It was conducted entirely in situ on a Sun Sunfire-880 with 8Gb of main memory and 4 750-MHz CPUs, which afforded the luxury of a complete factorial design with a large number of replications (1000).

The response measured was the cpu time (msec) required to recognize the randomly generated sequences corresponding to plans with varying characteristics. Note that cpu time was exclusive of any time used by the operating system or by other processes on the computer.

The factors and levels are summarized below:

Factor	Description	Levels
Order	The type of order constraints between plan nodes.	total, one, partial, unord
Depth	The level of plan depth	3, 4
Roots	The number of root goals in the plan library	10, 100, 1000

For each experimental condition, a single plan library was generated. Each such plan library had a number of features including the three tested factors.

1. Order (tested factor): This is an indication of how many and what type of ordering constraints there were between the actions in the methods in the plan library.
   - Total: the actions are linearly ordered with each action having a single ordering constraint with the action that precedes it in the initial definition.
   - One: all of the actions are ordered after the first action listed in the definition. However they are all unordered with respect to any action other than the first.
   - Partial: Each action (other than the first in the definition) may have one ordering constraint. This constraint orders the action after one of the randomly chosen earlier actions in the

definition. As such this means that methods can vary all the way from being totally ordered to completely un-ordered. This value for the ordering factor was specifically included in order to approximate real world plan libraries. In these cases the actions will be neither totally ordered or completely unordered.

- Unord: All of the actions are unordered with respect to each other.

2. Plan Library Depth (tested factor): This is a measure of the depth of the plan trees. In these plan trees "or nodes" (choice points) and "and nodes" (method expansions) alternate levels. In all cases the root is defined as an "or node" and levels alternate as they go down. For example in the case of depth three trees the root is an "or node" followed by a layer of "and nodes" followed by leaf nodes. The depth four trees have another layer of "or nodes" before the leaves.
3. Number of Roots (tested factor): This measures the number of root nodes in the plan library at 10, 100 and 1000 roots respectively.
4. Method Branching factor: This determines the number of actions at an "and node" (method definition). In all cases this was fixed at 4.
5. Choice Branching factor: This determines the number of actions at an "or node" (choice node). In all cased this was fixed at 3.

Note that all the actions in the plan libraries were unique. Thus, once an action is observed there is actually no ambiguity about what root intention the action must contribute to. This does not rule out the possibility of more than one instance of a given plan. However, this does allows us to make several inferences about the effect of various factors on the algorithm's runtime. We will return to discuss this later.

For each such plan library/experimental condition, 1000 test cases were generated. To generate a test case three unique roots were selected at random. For each of these roots a legal linearization of a plan for that root was generated, by choosing a single sub-action for "or nodes" and choosing a legal linearization for ordering constraints for "and nodes." Once a plan was generated for each of the three roots they were then randomly interleaved maintaining the ordering constraints of the individual plans but mixing the actions.

For each of the 1000 trial points, the internal clock was started, and PHATT was presented with the generated action sequence; after processing the sequence PHATT computed the probability distribution over the root goals. At this point, the clock was halted and the cpu time measured. This time was recorded for the condition. The files containing the cpu times (and other statistics) were parsed and combined with custom Perl programs and imported into S-Plus for charting and data analysis.

Overview of the Data

The entire ensemble of data (24,000 points) is charted below against all three factors in a trellis chart. Each narrow column of points represents an experimental condition. This chart is best viewed in COLOR. On the vertical axis, we have used a log scale for cpu time for compression reasons, since it spans 5 orders or magnitude. Each row of panels has the same $\log_{10}$ of cpu time, with 1 msec added to deal with those data points that registered as '0'.

The horizontal axis is the number of roots at equally spaced log units: 10, 100, 1000.

The left column of panels is for depth 3 plans, and the right column of panels is for depth 4 plans. The rows represent different order constraints. From the top down: unordered, total, partial, one.

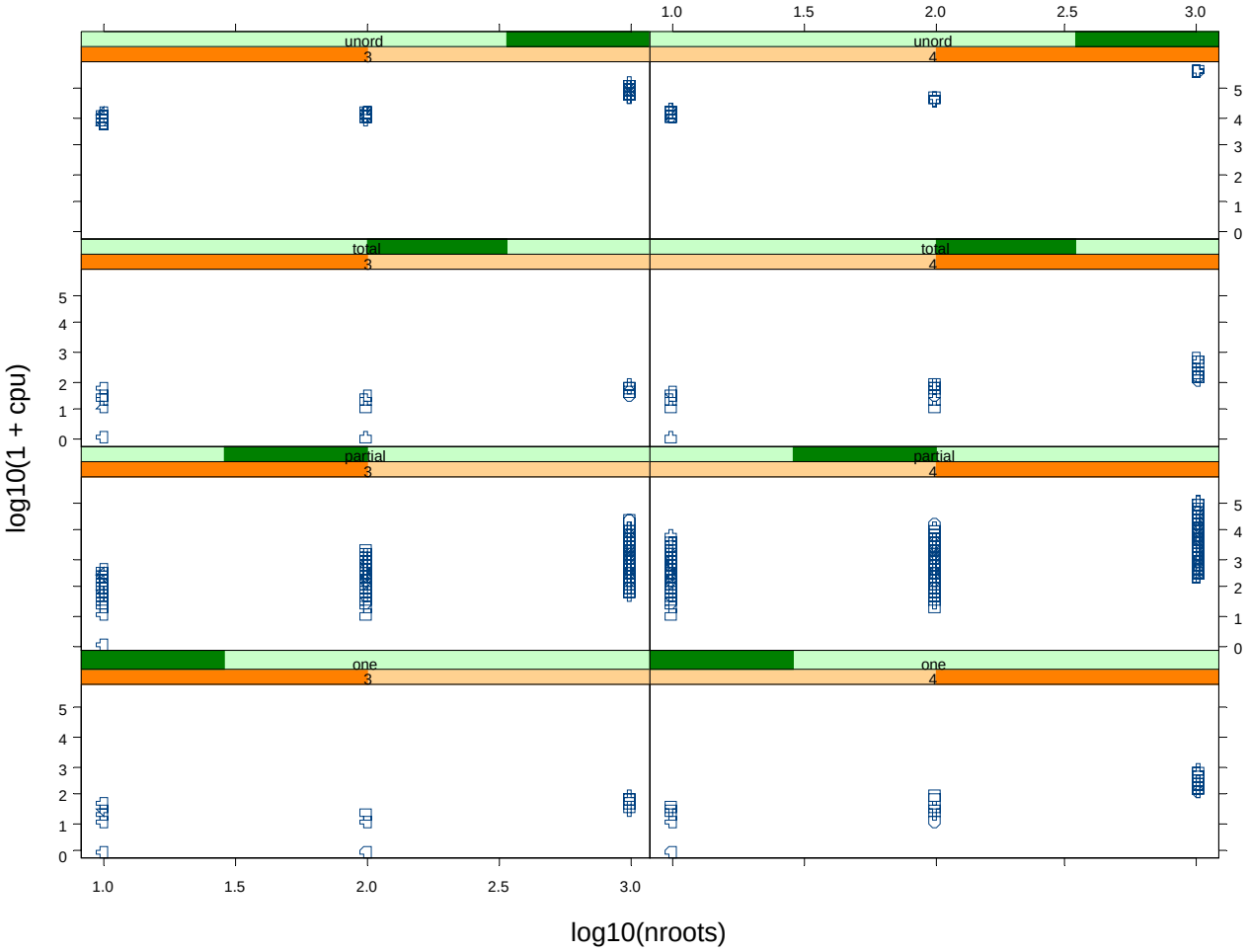

Figure 14. Trellis Chart

Comments

Several things are evident from this graph. The type of order constraint has a profound effect, both on means and on variance. Unordered plans exhibit the highest means; partially ordered plans also have relatively high means, but show the highest variance, regardless of the other factors. The difference between order "one" and order "total" are not so obvious. The number of roots seems to have the expected effect (increasing) on cpu time, but the size of this effect is less than we might have anticipated (recall these are log units). The plan depth (3 or 4) likewise seems to have the expected effect (increasing) on cpu time.

ANOVA Model

While most of the interesting conclusions are evident in the graph and not in doubt, an analysis of variance was conducted. After examination of residuals from the saturated model, it was clear that

the assumption of normality and uniform variance is doubtful. After examining some transforms, we found that this problem was substantially (though not completely) ameliorated by the log transform of the response, much as was done in the previous multipanel chart (See Section 4.2).

Model - Anova Table

```
Short Output:
Call:
   aov(formula = log(1 + cpu) ~ order * depth * roots, data = all)

Terms:
                   order    depth    roots order:depth order:roots depth:roots
 Sum of Squares 219167.2   7653.9  35494.6       148.3       167.1       348.0
Deg. of Freedom        3        1        2           3           6           2
                order:depth:roots Residuals
 Sum of Squares             263.1   13233.1
Deg. of Freedom                 6     23976

Residual standard error: 0.7429193
Estimated effects are balanced

                     Df Sum of Sq  Mean Sq  F Value Pr(F)
            order     3  219167.2 73055.73 132364.4     0
            depth     1    7653.9  7653.95  13867.6     0
            roots     2   35494.6 17747.31  32155.1     0
      order:depth     3     148.3    49.44     89.6     0
      order:roots     6     167.1    27.85     50.5     0
      depth:roots     2     348.0   174.00    315.3     0
order:depth:roots     6     263.1    43.86     79.5     0
        Residuals 23976   13233.1     0.55
```

We note that all main effects and interactions register as significant. The large number of replicates here means even quite small effects will be detectable.

Contrasts of the Order Factor

Of significant interest to us was if there really is a difference between 'total' and 'one' levels of *order*, or for that matter, between 'partial' and 'unord'. The Tukey HSD method was used to test these contrasts. The results imply significance at the 0.01 level for all the pairwise comparisons, with the ‘one’ level being slightly more time consuming for the algorithm as expected:

```
#  All of the diffs that are greater than lwr are significant.
#
> TukeyHSD(large.logaov, "order", ordered = TRUE, conf.level=0.01)
  Tukey multiple comparisons of means
    1% family-wise confidence level
    factor levels have been ordered

Fit: aov(formula = log(1 + cpu) ~ order * depth * roots, data = large)

$order
                    diff       lwr       upr
one-total     0.1748367 0.1706773 0.1789961  *
partial-total 2.4839996 2.4798403 2.4881590  *
unord-total   7.4876435 7.4834841 7.4918029  *
partial-one   2.3091629 2.3050036 2.3133223  *
unord-one     7.3128068 7.3086474 7.3169662  *
unord-partial 5.0036438 4.9994845 5.0078032  *
```

The story is more complicated than suggested by this test, since the interactions in the model are significant—a more complete analysis would look at fixed combinations of the other factors.

Analysis, Conclusions, and Future Questions for Experimentation

Several things are evident from this set of experiments.

1. The feature of the plan library that has the most significant effect on the algorithm's runtime is the ordering constraints within the plan library, followed by the number of roots in the plan library, followed by the actual depth of the plan trees.
2. It is not simply the number of ordering constraints that is important to runtime. Since 'total' and 'one' have the same overall number of constraints but 'one' has a higher average runtime we can conclude that the way in which the constraints are organized has a significant impact on the algorithms runtime.
3. The data suggests that the average runtime for the algorithm is scaling linearly in the number of roots in the plan library. This is very good news indeed for the PHATT algorithm
4. The runtime data for the 'partial' ordering condition fell consistently between the 'one' and 'unord' cases. This confirms our hypothesis that plan libraries without ordering constraints represents an upper bound or worst case for the ordering factor. This can be used not only in future experiments but also begins to set boundaries for our understanding of the algorithms performance.

There are a number of other experiments that should be performed to provide still more information about the algorithms performance.

1. Running another ordering condition that represents the opposite of 'one' but still maintains the same number of ordering constraints. In this case the last action of the plan would be ordered after all of the other actions in the plan. Intuition suggests that this value for the ordering factor should produce runtimes between 'one' and 'unord', this would provide still more evidence that the number of ordering constraints is not the critical feature but rather how they are constraining the plans in the library.
2. We need to do more tests at points between 100 and 1000 plan root nodes and even beyond 1000 in order to confirm the hypothesis that the algorithm is scaling linearly with the number of root intentions in the plan library
3. We need to do more test plan libraries with greater depth. Intuition suggests that the algorithm should scale exponentially in the depth of the plan trees. This results from the larger number of plans as a result of the number of layers of "or nodes" goes up. However even if this is born out by our experiments, this should not be seen as a problem since the depth of hierarchical plans in most real world applications is limited to a relatively small value. (i.e less than 10)
4. Further analysis is needed to understand why the variance of the algorithm is increasing as the number of root's increases. This is common in many computer programs. Our current hypothesis is that this is a result of some small linear searches embedded in the algorithm that are linked to the number of roots in the plan, however this has yet to be confirmed.

Post run examination of those cases that had particularly bad runtimes and introspection about the algorithm has also lead to a very important conclusion and suggested a set of experiments that

should be run. We have found that those data points with particularly long runtimes for the 'unord' case had a large number of possible explanations for the observations. That is since the 'unord' cases had no ordering constraints when it saw three actions that could contribute to a single plan, it could not rule out the possibility that each of these actions contributed to a separate instance of the root goal. That is the system was forced to consider the possibility that multiple instance of the same goal were being performed because the actions were unordered.

The hypothesis of the critical nature of the number of possible explanations is given weight by our results with 'one' and 'total'. Remember that all of the actions in the plan library are unique. This means that for the 'one' and 'total' cases there was in fact only a single possible explanation for the observed actions due to the ordering constraints. As a result, the algorithm was never required to consider the possibility that the agent was pursuing multiple instances of the same plan. In each of these cases, each plan had a unique first action. If the system does not see it then it doesn't have to consider the possibility of multiple instances of the root intention.

This hypothesis would suggest that the number and type (early in the plan vs. late in the plan) of ordering constraints in the plan library is a method of controlling the more important factor in the algorithm's runtime, namely the number of possible explanations for the observed actions. This corresponds to our intuitions that as the system is forced to consider a larger number of possible explanations for the observations that the runtime of the algorithm should increase. As a result we believe that one of the most important experiments that needs to be run is to control the number of possible explanations within a fixed size plan library and see the impact that it has.

Conclusions and Future Directions

Our project has achieved all of its major objectives. It has demonstrated that the skeptical systems approach to security is viable for limited plan libraries covering applications like the contract management system we have used. Further our experiments give us good reason to believe that the critical PHATT intent recognition algorithm is scalable to whole real world applications.

However our application of this technology to this domain has highlighted some limitations in the intent recognition algorithm that would be a significant asset in future research work and application to real world domains. For example, PHATT:

1. does not effectively handle plans with looping,
2. has only a qualitative temporal model and would benefit from a quantitative model,
3. could benefit from a typing system for plan variables,

Finally, more work should be done to understand those factors that impact the size of the explanation space captured by a plan library and the effect this has on the algorithm's runtime.

References

Geib, C.W., and Goldman, R.P., "Probabilistic Plan Recognition for Hostile Agents," *Proceedings of the FLAIRS01 Conference*, 2001a.

Geib, C.W. and Goldman, R.P., "Plan Recognition in Intrusion Detection Systems," *Proceedings of the DISCEX-II Conference*, 2001b.

Goldman, R.P., Geib, C.W., and Miller, C.A., "A New Model of Plan Recognition," In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, Stockholm, 1999.

Jensen, Finn V. *An Introduction to Bayesian Networks.* Springer: 1996.